```
for (i=left ; i<=right i++)
        arr[i] = temp[i]
    return inv=count;
}
int      mergesort (int arr[], int arr_size){
        int temp (arr,size)
        return merge-sort count (arr, temp, 0, arr_size)
}
int merge count ( int arr[], int temp[], int left
    int right) {
    int mid, inv=0;
    if (right > left) {
            mid = (right + left)/2
            inv+= merge-count (arr, temp, left mid)
        inv+ = merge count (arr, temp, mid, right)
        int = mergecount (arr, temp, left, right)
    }
    return inv;
}
```

10 Worst case :- $\Theta(n^2) - O(n^2)$

→ The worst case occurs when the partition proception picks up greatest or smallest element as pivot

Best case :- $\Theta(n\log n) - \Omega(n\log n)$

→ The best case occur when the partition process a picks the middle elent as pivot

→ Has running time of $O(n^2)$ that makes it subceptible in application.
→ Most often than not runs of $O(n \log n)$
→ High space efficiency by executing in place
→ C sort utility in C programming language is powered quicksort.

Q Inversion count for an array indicates how far the array being sort if array sorted, then the inversion 0, but if the array is sorted in the reverse order the count in the maximum.

```
int merge(int arr[], int temp[], int left, int mid, int right)
    int i,j,K;
    int inv_count = 0;
    i = left;   j = mid,   k = left;
    while ((i <= mid-1) && (j <= right)){
        if (arr[i] < arr[j]{
                temp[K++] = arr[i++];
        }
        else {
                temp[K++] = arr[j++];
                inv_count = inv_count + mid-i);
        }
}
    while (i <= mid-1)
            temp[K++] = arr[i++]
    while ( j <= right
            temp[K++] = arr[j++]
```

(6) Time Complexity & Recurrence relation

Best case = O(1)
Average = O(nlogn)
Worst = T(n) = T(n/2)+c

$$T(n) \begin{cases} c & \text{if } i >= f \\ T(n/2)+c & \text{otherwise.} \end{cases}$$

(7) Recursive :-

```
int binary (int arr[], int low, int high, int x)
{
    if (low > high)
    {
        return -1;
    }
    int mid = (low+high)/a
    if (x == arr[mid])
        return mid
    else if (x < arr[mid])
        return binary (arr, low, mid-(1U));
    else
        return binary (arr, mid+1, high, x);
}
```

8 Quick sort is best in terms of practical use as, most wide sorting algorithm at present

```
if(a[j] > a[j+1])
    temp = a[j]
    a[j] = a[j+1]
    a[j+1] = temp
  endif
 end for
end for
```

(3)

Algorithm	Time Complexity		
	Best	Average	Word.
Selection Sort	$\Omega(n^2)$	$\theta(n^2)$	$O(n^2)$
Bubble Sort	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$
Insertion Sort	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$
Heap Sort	$\Omega(n \log n)$	$\theta(n \log n)$	$O(n \log n)$
Quick Sort	$\Omega(n \log n)$	$\theta(n \log n)$	$O(n^2)$
Merge Sort	$\Omega(n \log n)$	$\theta(n \log n)$	$O(n \log n)$

(4) Bubble Sort, insertion sort, Selection Sort in place Souting algo.

- Merge Sort is a stable but not an implace Algorithm
- Insertion Sort is also used for Online Sorting
- Quick Sort is not stable but is an in-place Algo
- Heap Sort is an in-place but not stable
- Selection Sort is also an Online Sorting Algo.

for For $i=0$ to $n-1$ do:
small sup $= i$
For $j = 0+1$ to $n-1$ do'
if $A(j) < A(\text{smallsub})$
smallsub $= j$
end if
end afor
$temp = A[i]$
$A[i] = A[\text{smallsub}]$
$A[\text{smallsub}] = temp$
end for

Insertion Sout. →
For $i=1$ to $n-1$
$j=1$
do while $(j>0)$ and $(a[j]<a[j-1])$
$temp = a[j]$
$a[j] = a[j-1]$
$a[j-1] = temp$
$j = j-1$
end do
end for.

Bubble Sort →

for $i=0$ to $n-2$
for $j=0$ to $n-2$
~~for $j=0$ to~~

DAA Tutorial - 3

NAME = TARUN KUMAR DHIMAN
ROLL No = 1918017
Sec = IT

① Search (arr, n, x)
if (arr[n-1] == x)
{
return "found";
}
int backup = arr[n-1]
arr[n-1] = x;
for (int i=0; ; i++
{
if (arr[i] == x)
{
arr[n-1] = backup
if (i<n-1)
return "found";
else
return "Notfound";
}
}

② Selection Sort;